CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #11: Query Optimization
Some parts from (a copy of the paper is on the course webpage)

Select *
From Blah B
Where B.blah = blah

Usually there is a heuristics-based rewriting step before the cost-based steps.
Multiple Algorithms: Range Searches

- Sequential Scan
- Hashes
- B-Trees
- ....

- Saw some of them in previous lectures
Multiple Algorithms: Joins

- Merge-Join (like merge-sort)
- Hash-Join (using hashes)
- Indexed-Join (using indexes)
- Nested loops Join (most obvious)
- ....

- Saw some of them in previous lectures
Why Query optimization?

- SQL: ~declarative
- good q-opt -> big difference
  - eg., seq. Scan vs
  - B-tree index, on P=1,000 pages
Q-opt steps

- bring query in internal form (eg., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best
Q-opt - example

```
select name
from STUDENT, TAKES
where c-id = '4604' and
STUDENT.ssn = TAKES.ssn
```
Q-opt - example

select name
from STUDENT, TAKES
where c-id = '4604' and
STUDENT.ssn = TAKES.ssn

Join Predicate => STUDENT.ssn = TAKES.ssn
(is assumed to be part of the join)

Non-join Predicate => c-id = '4604'
(part of the explicit selection)
Q-opt - example

STUDENT \(\pi\) TAKES

STUDENT \(\sigma\) TAKES

Canonical form
Q-opt - example

Canonical Form has the following properties:
1. Push Selections as much as possible.
2. Push Projections as much as possible
3. It is a left-deep join tree (we will see this later)
Q-opt - example

Hash join; merge join; nested loops;

\[ \pi \]

\[ \sigma \rightarrow \text{Index; seq scan} \]

STUDENT

TAKES
Equivalence of expressions

- A.k.a.: syntactic q-opt
- in short: perform selections and projections early
Equivalence of expressions

Q: How to prove a transformation rule?

\[ \sigma_P(R1 \Join R2) = \sigma_P(R1) \Join \sigma_P(R2) \]

A: use RA, to show that LHS = RHS, eg:

\[ \sigma_P(R1 \cup R2) = \sigma_P(R1) \cup \sigma_P(R2) \]

Prakash 2016
Equivalence of expressions

\[ \sigma_p(R1 \cup R2) = \sigma_p(R1) \cup \sigma_p(R2) \]

\[ t \in LHS \iff \]

\[ t \in (R1 \cup R2) \land P(t) \iff \]

\[ (t \in R1 \lor t \in R2) \land P(t) \iff \]

\[ (t \in R1 \land P(t)) \lor (t \in R2 \land P(t)) \iff \]
Equivalence of expressions

\[ \sigma_p(R1 \cup R2) = \sigma_p(R1) \cup \sigma_p(R2) \]

...  

\[(t \in R1 \land P(t)) \lor (t \in R2 \land P(t)) \iff (t \in \sigma_p(R1)) \lor (t \in \sigma_p(R2)) \iff t \in \sigma_p(R1) \cup \sigma_p(R2) \iff t \in \text{RHS} \]

QED
Equivalence of expressions

Q: how to disprove a rule??

\[ \pi_A (R1 - R2) = \pi_A (R1) - \pi_A (R2) \]

Construct a counter-example!
Equivalence of expressions

- **Selections**
  - perform them early
  - break a complex predicate, and push
    \[ \sigma_{p_1 \land p_2 \land \ldots \land p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots \sigma_{p_n}(R))\ldots) \]
  - simplify a complex predicate
    - (‘X=Y and Y=3’) -> ‘X=3 and Y=3’
Equivalence of expressions

- Projections
  - perform them early (but carefully...)
    - Smaller tuples
    - Fewer tuples (if duplicates are eliminated)
  - project out all attributes except the ones requested or required (e.g., joining attr.)
Equivalence of expressions

- Joins
  - Commutative, associative
    
    \[
    R \bowtie S = S \bowtie R
    \]
    
    \[
    (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)
    \]
  - Q: n-way join - how many diff. orderings?
Equivalence of expressions

- Joins - Q: n-way join - how many diff. orderings?
- A: Catalan number $\sim 4^n$
  - Exhaustive enumeration: too slow.
(Some) Transformation Rules (1)

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.
   \[ \sigma_{\theta_1 \land \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E)) \]

2. Selection operations are commutative.
   \[ \sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E)) \]

3. Only the last in a sequence of projection operations is needed, the others can be omitted.
   \[ \Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{L_n}(E))\ldots)) = \Pi_{L_1}(E) \]

4. Selections can be combined with Cartesian products and theta joins.
   a. \[ \sigma_{\theta}(E_1 \times E_2) = E_1 \times_{\theta} E_2 \]
   b. \[ \sigma_{\theta_1}(E_1 \times_{\theta_2} E_2) = E_1 \times_{\theta_1 \land \theta_2} E_2 \]
5. Theta-join operations (and natural joins) are commutative.
\[ E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1 \]

6. (a) Natural join operations are associative:
\[ (E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3) \]

(b) Theta joins are associative in the following manner:
\[ (E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \land \theta_3} E_3 = E_1 \bowtie_{\theta_1 \land \theta_3} (E_2 \bowtie_{\theta_2} E_3) \]

where \( \theta_2 \) involves attributes from only \( E_2 \) and \( E_3 \).
(Some) Transformation Rules (3)

7. The selection operation distributes over the theta join operation under the following two conditions:
   (a) When all the attributes in $\theta_0$ involve only the attributes of one of the expressions ($E_1$) being joined.

   $$\sigma_{\theta_0}(E_1 \Join_\theta E_2) = (\sigma_{\theta_0}(E_1)) \Join_\theta E_2$$

   (b) When $\theta_1$ involves only the attributes of $E_1$ and $\theta_2$ involves only the attributes of $E_2$.

   $$\sigma_{\theta_1 \land \theta_2}(E_1 \Join_\theta E_2) = (\sigma_{\theta_1}(E_1)) \Join_\theta (\sigma_{\theta_2}(E_2))$$
Q-opt steps

- bring query in internal form (eg., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best
Cost-based Query Sub-System

Queries

- Select *
  From Blah B
  Where B.blah = blah

Usually there is a heuristics-based rewriting step before the cost-based steps.

Query Parser

Query Optimizer

- Plan Generator
- Plan Cost Estimator

Query Plan Evaluator

Catalog Manager

- Schema
- Statistics

Prakash 2016
Cost estimation

- Eg., find ssn’s of students with an ‘A’ in 4604 (using seq. scanning)
- How long will a query take?
  - CPU (but: small cost; decreasing; tough to estimate)
  - Disk (mainly, # block transfers)
- How many tuples will qualify?
- (what statistics do we need to keep?)
Cost estimation

- Statistics: for each relation ‘r’ we keep
  - nr : # tuples;
  - Sr : size of tuple in bytes
Cost estimation

- Statistics: for each relation ‘r’ we keep
  - …
  - \( V(A,r) \): number of distinct values of attr. ‘A’
  - (recently, histograms, too)
Derivable statistics

- blocking factor = max# records/block (=??)
- br: # blocks (=??)
- SC(A,r) = selection cardinality = avg# of records with A=given (=??)

Prakash 2016
VT CS 4604
Derivable statistics

- blocking factor = max# records/block (= B/Sr ; B: block size in bytes)
- br: # blocks (= nr / (blocking-factor) )
**Derivable statistics**

- $\text{SC}(A,r) = \text{selection cardinality} = \text{avg\# of records with } A=\text{given} \ ( = \text{nr} / V(A,r) ) \ (\text{assumes uniformity...})$

eg: 10,000 students, 10 departments – how many students in CS?
Additional quantities we need:

- For index ‘i’:
  - $f_i$: average fanout ($\sim$50-100)
  - $H_{Ti}$: # levels of index ‘i’ ($\sim$2-3)
    - $\sim \log(#\text{entries})/\log(f_i)$
  - $L_{Bi}$: # blocks at leaf level
Statistics

- Where do we store them?
- How often do we update them?
Q-opt steps

- bring query in internal form (eg., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best
Selections

- we saw simple predicates (A=constant; eg., ‘name=Smith’)
- how about more complex predicates, like
  - ‘salary > 10K’
  - ‘age = 30 and job-code=“analyst”’
- what is their selectivity?
Selections – complex predicates

- selectivity \( \text{sel}(P) \) of predicate \( P \):
  - \( \text{\textasciitilde} = \) fraction of tuples that qualify
  - \( \text{sel}(P) = \text{SC}(P) / \text{nr} \)
Selections – complex predicates

- eg., assume that $V(\text{grade}, \text{TAKES}) = 5$ distinct values
- simple predicate $P$: $A = \text{constant}$
  - $\text{sel}(A = \text{constant}) = 1/V(A,r)$
  - eg., $\text{sel}(\text{grade} = 'B') = 1/5$
- (what if $V(A,r)$ is unknown??)
Selections – complex predicates

- range query: \( \text{sel}( \text{grade} \geq 'C') \)
  - \( \text{sel}(A > a) = \frac{(A_{\text{max}} - a)}{(A_{\text{max}} - A_{\text{min}})} \)

![Diagram of grade count](image)
Selections - complex predicates

- negation: sel( grade != 'C')
  - sel( not P) = 1 – sel(P)
  - (Observation: selectivity =~ probability)
Conjunction:

- `sel( grade = ‘C’ and course = ‘4604’ )`
- `sel(P1 and P2) = sel(P1) * sel(P2)`
- INDEPENDENCE ASSUMPTION
Selections - complex predicates

- **Disjunction:**
  - \( \text{sel( grade = 'C' or course = '4604') } \)
  - \( \text{sel(P1 or P2) = sel(P1) + sel(P2) - sel(P1 and P2)} \)
  - \( \text{= sel(P1) + sel(P2) - sel(P1)*sel(P2)} \)
  - INDEPENDENCE ASSUMPTION, again
Selections - complex predicates

- disjunction: in general
  - \( \text{sel}(P_1 \textbf{or} P_2 \textbf{or} ... P_n) = \\
  \quad 1 - (1 - \text{sel}(P_1)) \times (1 - \text{sel}(P_2)) \times ... \times (1 - \text{sel}(P_n)) \)
Selections Selectivity – summary

- sel(A=constant) = 1/V(A,r)
- sel(A>a) = (A_{max} – a) / (A_{max} – A_{min})
- sel(not P) = 1 – sel(P)
- sel(P1 and P2) = sel(P1) * sel(P2)
- sel(P1 or P2) = sel(P1) + sel(P2) – sel(P1)*sel(P2)
- sel(P1 or ... or Pn) = 1 - (1-sel(P1)) * ... * (1-sel(Pn))

- UNIFORMITY and INDEPENDENCE ASSUMPTIONS
Result Size Estimation for Joins

Q: Given a join of R and S, what is the range of possible result sizes (in #of tuples)?
   – Hint: what if R_cols ∩ S_cols = ∅?
   – R_cols ∩ S_cols is a key for R (and a Foreign Key in S)?
Result Size Estimation for Joins

- General case: $R_{cols} \cap S_{cols} = \{A\}$ (and A is key for neither)
  - match each R-tuple with S-tuples
    \[
    \text{est}\_\text{size} \approx NTuples(R) \times NTuples(S) / NKeys(A,S)
    \]
  - symmetrically, for S:
    \[
    \text{est}\_\text{size} \approx NTuples(R) \times NTuples(S) / NKeys(A,R)
    \]
  - Overall:
    \[
    \text{est}\_\text{size} = \frac{NTuples(R) \times NTuples(S)}{\max\{NKeys(A,S), NKeys(A,R)\}}
    \]
On the Uniform Distribution Assumption

- Assuming uniform distribution is rather crude

Distribution D

Uniform distribution approximating D
Histograms

- For better estimation, use a histogram

Equidepth histogram ~ quantiles

Equiwidth histogram
Q-opt Steps

- bring query in internal form (e.g., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- **generate alt. plans**
  - single relation
  - multiple relations
- estimate cost; pick best
- Selections – eg.,
  
  ```sql
  select *
  from TAKES
  where grade = 'A'
  ```

- Plans?
plan generation

- Plans?
  - seq. scan
  - binary search
    - (if sorted & consecutive)
  - index search
    - if an index exists
plan generation

seq. scan – cost?
- br (worst case)
- br/2 (average, if we search for primary key)
plan generation

binary search – cost?
if sorted and consecutive:
- ~log(br) +
- SC(A, r)/fr (=blocks spanned by qual. tuples)
plan generation

estimation of selection cardinalities SC(A,r):
– we saw it earlier how to do it for general conditions
method #3: index – cost?

- Roughly $\log(N)$, but exact cost tricky
Q-opt Steps

- bring query in internal form (eg., parse tree)
- ... into ‘canonical form’ (syntactic q-opt)
- generate alt. plans
  - single relation
  - multiple relations
- estimate cost; pick best
n-way joins

- $r_1 \text{ JOIN } r_2 \text{ JOIN } ... \text{ JOIN } r_n$
- typically, break problem into 2-way joins
  - choose between NL, sort merge, hash join, ...
Queries Over Multiple Relations

- As number of joins increases, number of alternative plans grows rapidly → need to restrict search space
- Fundamental decision in System R (IBM): only left-deep join trees are considered. Advantages?
Queries Over Multiple Relations

- As number of joins increases, number of alternative plans grows rapidly → need to restrict search space
- Fundamental decision in System R (IBM): only left-deep join trees are considered. Advantages?
  - fully pipelined plans.
    - Intermediate results not written to temporary files.
Queries over Multiple Relations

- Enumerate the orderings (= left deep tree)
- enumerate the plans for each operator
- enumerate the access paths for each table

Dynamic programming, to save cost estimations
(we wont cover exact algorithm in class)
1. Enumerate relation orderings:

Prune plans with cross-products immediately!
2. Enumerate join algorithm choices:

+ do same for 4 other plans

→ 4*4 = 16 plans so far..
SELECT  S.sname, B.bname, R.day
FROM   Sailors S, Reserves R, Boats B

3. Enumerate **access method** choices:

```
NLJ  
/    \
R    B
S
```

```
NLJ  
/    \
R    B
S
```

```
NLJ  
/
  \
R
S
```

```
NLJ  
/    \
R    B
S
```

+ do same for other plans

```
NLJ   
/     \
R     B
S
```

```
NLJ  
/
  \
R
S
```

```
NLJ   
/     \
R     B
S
```

```
NLJ  
/
  \
R
S
```

```
NLJ   
/     \
R     B
S
```

(heapscan)

(INDEX scan on R.bid)
Now estimate the cost of each plan

Example:

\[
\text{NLJ} \quad \text{NLJ} \quad \text{B} \\
\text{S} \quad \text{R} \\
\text{(heap scan)} \quad \text{(heap scan)} \quad \text{(INDEX scan on R.sid)}
\]
Conclusions

- Ideas to remember:
  - canonical parse tree
  - syntactic q-opt – do selections/projections early
    - More complicated rules are also used
  - How to get selectivity estimations (uniformity, independence)
    - We saw mainly range and equality predicates
    - More complicated: histograms; join selectivity
  - left-deep joins
    - dynamic programming